YKSIKKÖTESTAUKSEN HYÖDYNTÄMINEN MICROSOFT DYNAMICS AX:SSA

Pasi Piponius

Opinnäytetyö
Maaliskuu 2011
Tietotekniikka
Ohjelmistotekniikka
Tampereen ammattikorkeakoulu

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikka
Ohjelmistotekniikka

PIPONIUS, PASI: Yksikkötestauksen hyödyntäminen Microsoft Dynamics AX:ssa

Opinnäytetyö 28 s.
Maaliskuu 2011

Tässä opinnäytetyössä perehdytään ohjelmistotestaukseen Microsoft Dynamics AX 2009 toiminnanohjausjärjestelmää koskien. Tarkennettuna tutkimuskohteena on yksikkötestaus, jonka käyttöä analysoidaan kyseisessä toiminnanohjausjärjestelmässä.

Tavoitteena on myös saada yleiskatsaus Dynamics AX 2009:n toiminnasta, kuvata yksikkötestauksen sisältöä, hyötyjä ja haasteita. Työ on koottu erinäisistä kirjoista sekä muista materiaaleista, joissa on keskeisimmät asiat Microsoft Dynamics AX 2009:iin liittyen ja joista saa kuvan sen toiminnasta sekä yksikkötestin perustamisesta.

Tästä työstä selviää miten yksikkötestaus toteutetaan ja mitä sen sisällä tapahtuu. Työssä annetaan myös esimerkkinä yksinkertainen testitapaus, josta näkee yksikkötestauksen käytännön hyödyn. Testitapauksessa verrataan suoritusaikoja dimensioiden ja virheiden kasvaessa.

Asiasanat: Yksikkötesti, Ohjelmistotekniikka

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University Of Applied Sciences
Computer technology training program
Software orientation

PIPONIUS, PASI: Taking advantage of unit testing in Microsoft Dynamics AX

Bachelor's thesis 28 pages
March 2011

This thesis explores software testing associated to Microsoft Dynamics AX 2009 Enterprise Resource Planning -system. The specified research target is unit testing and the use of it will be analyzed regarding to before mentioned ERP system.

The goal of this thesis is to acquire a comprehensive overview of Dynamics AX 2009's functionalities, to describe the content of unit-testing and to clarify its benefits and challenges. The relevant information for this thesis is gathered from various books and other publications dealing with the key aspects of Microsoft Dynamics AX 2009 and creating unit-test cases.

This thesis presents how to perform an unit-test and what occurs within the test. A simple test case is also performed in order to illustrate the practical benefits of unit-testing. Time-based performance is compared in the test case by increasing the amount of dimensions and errors.

Keywords: Unit Testing, Software

SISÄLLYS

SANASTO

ERP	Enterprice Resource Planning eli toiminnanohjaus
Integraatio	Kahden tai useamman erillisen osan yhdistäminen yhdeksi kokonaisuudeksi
Moduuli	Itsenäinen osa, jollaisista voidaan koota erilaisia kokonaisuuksia
Layer	Dynamics AX:ssa on eri tasoja, joille toiminnallisuus voidaan tehdä
Form	Lomake
Syntaksi	Säännöstö kielen muodosta
Service	Palvelu
TDD	Test Driven Development eli testivetoinen kehitys
Framework	Ohjelmistokehys
Parametri	Ohjelmalle välitettävä tieto
Kommitointi	Tietokantaan vietyjen muutosten vahvistus
Metodi	Sisältää osan ohjelman toiminnallisuudesta
Transaktio	Joukko peräkkäin suoritettavia tapahtumia
Rajapinta	Ohjelmien välinen yhtymäkohta
Performanssi	Suorituskyky
Funktio	Olioiden välinen riippuvuussuhde
Iteraatio	Ohjelmistoprojektin kierros

1 JOHDANTO

Työ on tehty toimeksiantona Mepco Oy:lle, jossa ei ole aikaisemmin käytetty yksikkötestaustyökaluja toiminnanohjausratkaisuissa. Mepco Oy on operatiivisia yritysohjelmistoja toimittava ja integroiva tietotekniikkayhtiö, joka yhdistää talouden- ja toiminnanohjauksen, asiakkuudenhallinnan sekä henkilöstönohjauksen johtavat ohjelmistotuotteet vahvaan toimiala- ja sovellusalueosaamiseen sekä asiakaslähtöiseen paikalliseen tukeen. Mepcon edustamat tuotteet ovat mm. Microsoft Dynamics AX ja Microsoft Dynamics CRM. Näitä ohjelmistoja toimitetaan asiakasyrityksille projektimuotoisena siten, että implementointi onnistuu ja tuottaa asiakkaalle luvatut hyödyt.

Työn tarkoituksena on tutkia onko Mepco Oy:n mahdollista hyödyntää Microsoft Dynamics AX 2009:n yksikkötestaustyökalua kehityksessä ja selvittää yksikkötestin avulla kasvaako kirjauskansion rivien kirjausaika dimensioiden ja virheiden lisääntyessä. Dimensioihin tutustutaan tarkemmin kappaleessa 5.

Tässä työssä tutustutaan yrityskäyttöön kehitetyn Microsoft Dynamics AX 2009 toiminnanohjausjärjestelmän yksikkötestaukseen. Työn tarkoituksena on koota suomenkielistä tietoa yksikkötestauksesta, tutustua yksikkötestauksen toimintaan, sen parametreihin, sekä esittää esimerkki, jossa sitä voi hyödyntää. Esimerkkinä käytetään kirjausrivien kirjausta pienellä dimensiomäärällä ilman virheitä ja sitä verrataan suureen määrään, jossa virheitä ilmaantuu vanhentuneiden dimensioiden vuoksi. Kappaleissa kerrotaan myös Microsoft Dynamics AX 2009:n kehityksestä, ohjelmointikielestä sekä joistain toiminnallisuuksista. Kappaleessa 2 kerrotaan toiminnanohjausjärjestelmistä yleisesti sekä niiden toiminnasta. Kappaleessa 3 esitellään lyhyesti Microsoft Dynamics AX:n historia ja kehitys. Kappaleessa 4 tutustutaan tarkemmin testaukseen, testiluokkiin sekä Dynamics AX:n työkaluihin ja parametreihin. Kappaleessa 5 suoritetaan erittäin yksinkertainen testi, jolla saadaan kirjaustapahtumaan kulunut aika. Kappaleessa 6 on yhteenveto edellisistä kappaleista ja johtopäätöksiä työkalun soveltuvuudesta työelämään.

2 TOIMINNANOHJAUSJÄRJESTELMÄ

Toiminnanohjausjärjestelmä eli ERP-järjestelmä (Enterprise Resource Planning) on yrityksen tietojärjestelmä, jonka avulla pyritään parantamaan yrityksen toimintojen tehokkuutta niin toiminnallisesti kuin taloudellisestikin integroimalla samaan järjestelmään eri osastoja palvelevia toimintoja. (Wikipedia. Toiminnanohjausjärjestelmät. Luettu 3.1.2011)

Tällaisia yrityksen sisäisiä toimintoja ovat mm. tuotantoon, jakeluun, varastonhallintaan, laskutukseen ja kirjanpitoon liittyvät asiat. Olennaista toiminnanohjausjärjestelmän käytössä on sen reaaliaikaisuus. Kun kaikki tiedot tallennetaan samaan tietokantaan, tiedot saadaan sieltä nopeasti, jolloin päällekkäisen työn teon tarve vähenee, sillä kaikki päätöksentekoon tarvittava tieto on helposti ja nopeasti saatavissa yhdestä paikasta.

Tietokoneiden ja internetin arkipäiväistymisen seurauksena nykyään lähes jokaisessa yrityksessä on käytössä jonkinlainen toiminnanohjausjärjestelmä korvaamassa perinteisiä manuaalisesti tehtäviä toimintoja.

Toiminnanohjausjärjestelmän käyttöönotto yrityksessä on usein kallista ja aikaa vievää. Lisäksi monimutkaisten kokonaisuuksien hallinta vaatii runsaasti asiantuntemusta. Tästä syystä yhä useampi yritys on siirtynyt käyttämään valmiita ohjelmistoja, jotka nykypäivänä ovat myös helposti muunneltavissa, jolloin yritys voi itse päättää, mitkä moduulit toiminnanohjausjärjestelmästä ottaa käyttöön.

Kappaleessa kolme esitellään Microsoftin tarjoamaa valmisohjelmistoa Microsoft Dynamics AX:a, joka on laajalti kansainvälisesti käytetty toiminnanohjausjärjestelmä tällä hetkellä. Dynamics AX:n kilpailukyky perustuu sen laajaan muunneltavuuteen sekä siihen, että se tarjoaa hyvät integrointi- ja yhteensopivuusmahdollisuudet yritysten käytössä olevien järjestelmien kanssa.

3 MICROSOFT DYNAMICS AX

Microsoft Dynamics AX on toiminnanohjausjärjestelmä, joka kehitettiin kattamaan kaikki yrityksen osa-alueet, kuten talousasiat, asiakassuhteet, yrityspalvelut, projektinhallinta sekä henkilöstöasiat. Dynamics AX:n hyöty onkin siinä, ettei tarvita useita eri ohjelmia, vaan kaikki yrityksen asiat saadaan saman ohjelman sisälle, jolloin vältytään integraatio-ongelmilta sekä pitkiltä vasteajoilta. Järjestelmä on myös kansainvälinen, joten sen avulla hoituvat niin maailmanlaajuiset liiketoiminnan osa-alueet kuin kotimaisetkin. Moduulit näkyvät kuviossa 1. (Microsoft, Microsoft Dynamics AX; Wikipedia, Microsoft Dynamics AX. Luettu 4.1.2011)

Microsoft Dynamics AX tarjoaa myös mahdollisuuden räätälöidä ohjelman juuri omien tarpeiden mukaan. Ohjelmassa on useita eri layereitä, joiden avulla voidaan tehdä muutoksia ilman, että ne vaikuttavat muiden tasojen ominaisuuksiin. Jos esimerkiksi lomakkeesta puuttuu haluttu tieto, se voidaan lisätä sinne itse. Järjestelmä tukee 45 eri kieltä, joihin sisältyy myös suomen kieli.

Ohjelmisto perustuu olio-ohjelmointiin ja kyselyt kantaan tehdään SQL-lausekkeilla. Ohjelmointikielenä toimii X++, joka on varta vasten kehitetty Dynamics AX:aa varten. X++:n syntaksi on hyvin javamainen, joten siihen on helppo päästä sisälle, jos javaohjelmoinnin syntaksi on hallussa.

3.1 Historia ja kehitys

Microsoft Dynamics AX:n kehitys on ollut pitkä ja monivaiheinen prosessi, johon on kuulunut monta siirtymävaihetta. Siihen on tullut useita uusia toiminnallisuuksia, ominaisuuksia sekä parannuksia eri versionumeroiden myötä ja sitä kehitetään edelleen.

3.1.1 Historia

Microsoft Dynamics AX kehitettiin alun perin Tanskassa, mutta Microsoft osti sen vuonna 2002. Tuotteen suosio kasvoi nopeasti, minkä ansiosta Microsoft pystyy jatkuvasti jatkokehittämään sitä toiveiden mukaisesti.

Microsoftin saatua Dynamics AX:n haltuunsa, siitä on tullut kolme versionumeroa ja neljäs saapuu vuoden 2011 aikana. Versionumerot ovat Axapta 3.0, Dynamics AX 4.0, Dynamics AX 2009, joka alun perin nimettiin AX 5.0:ksi, sekä myöhemmin vuonna 2011 saapuva Dynamics AX 6. (Inside Microsoft Dynamics AX 2009, xxiv. Luettu 5.1.2011)

Nimen Axapta alkuperästä on monta hypoteesiä. Yksi niistä on, että ainoa vaatimus nimessä oli kirjain ”X” ja näin päädyttiin Axaptaan. (Inside Microsoft Dynamics AX 2009, xxv. Luettu 5.1.2011)

3.1.2 Kehitys

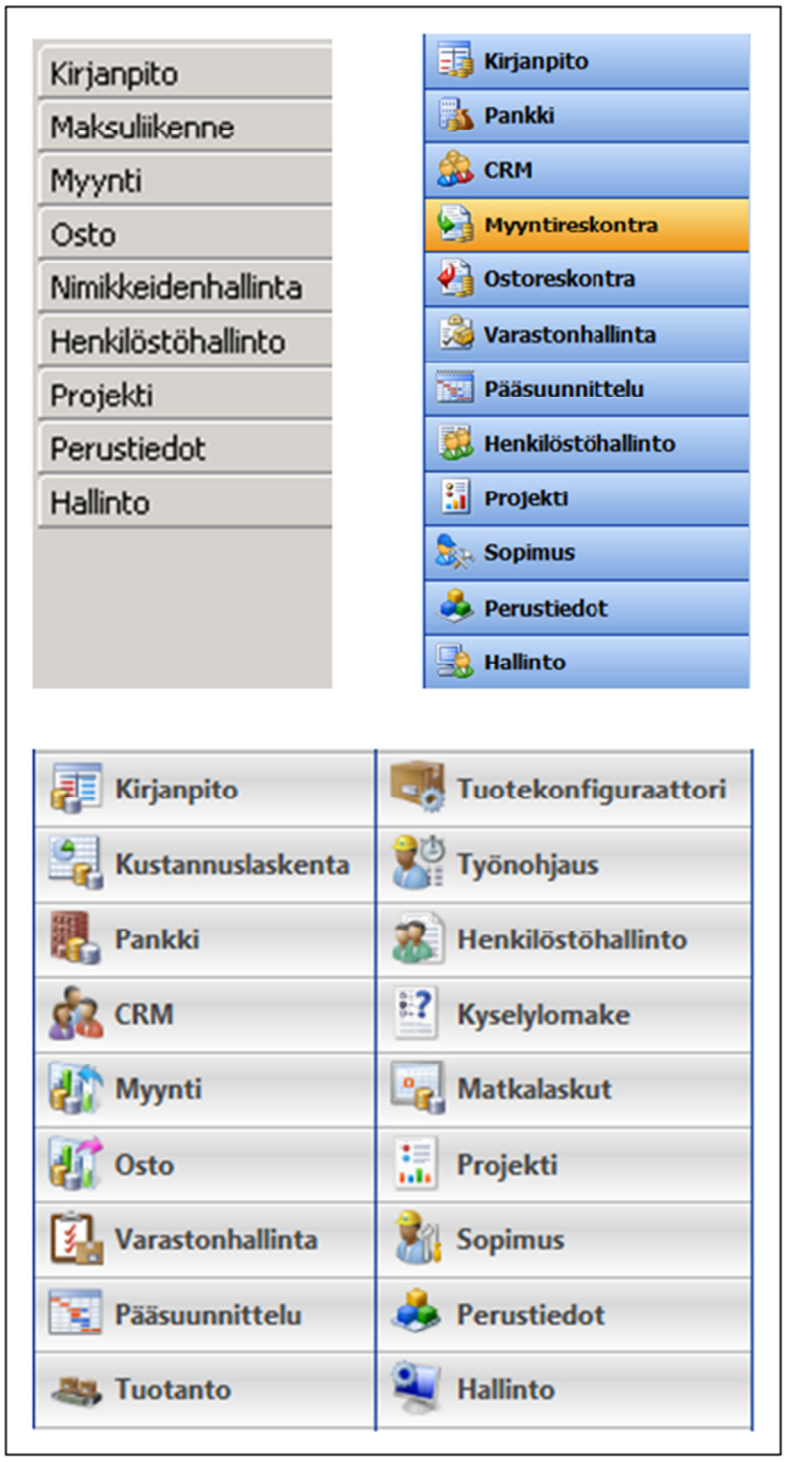
Microsoft lisäsi Axapta 3.0:aan muun muassa yritysportaalin, käyttäjäsuojan, järjestelmäkonfiguraation sekä kasvatti maayhteensopivuutta ja paransi tuotantotyökaluja. (Wikipedia, Microsoft Dynamics AX. Luettu 5.1.2011)

Dynamics AX 4.0:n mukana tuli uusi ulkoasu ja tämä oli ensimmäinen versio, jonka Microsoft kehitti alusta asti itse. Tämä mahdollisti sen, että Microsoft pystyi kehittämään Dynamics AX 4.0:n silmällä pitäen muita Microsoftin tekniikoita, jolloin yhteensopivuus parani ja järjestelmän suorituskyky kasvoi. Esimerkiksi AOS tuli Windowsin serviceksi.

Dynamics AX 2009:än paranneltiin vielä käyttöliittymää sekä ulkoasua. Muita suuria muutoksia olivat dimensiot, aikavyöhyketuki sekä yritysportaali, joka kehitettiin Visual Studiolla.

Uuden version eli Dynamics AX 6:n kerrotaan sisältävän jälleen uuden käyttöliittymän, parannuksia tiettyihin moduuleihin sekä koodieditorimuutoksia.

Kuviossa 1 on esitetty toiminnanohjausjärjestelmän kehittymistä Dynamics Axapta 3.0:sta Dynamics AX 2009:ään käyttöliittymän näkökulmasta. Eri versioita vertaillessa on myös huomattavissa sisältömoduulien määrän kasvu, joka on kaksinkertaistunut ensimmäisen ja viimeisimmän version välillä.



KUVIO 1. Käyttöliittymien ja moduulien kehitys. Vasemmalla ylhäällä Axapta 3.0, oikealla ylhäällä Dynamics AX 4.0 ja alhaalla Dynamics AX 2009.

4 OHJELMISTOTESTAUS

Ohjelmistotestauksen päätavoitteena on havaita ohjelmiston toimintakykyyn vaikuttavia häiriöitä, jolloin varmistetaan, että ohjelmisto toimii loppukäyttäjällä oikein kaikissa tilanteissa ja olosuhteissa niissä ympäristöissä, joissa ohjelmiston on tarkoitus toimia. Ohjelmistotestausta voisikin kuvailla suunnitelmalliseksi virheiden etsimiseksi ohjelmaa tai sen osaa suorittamalla. (Testauksen tavoitteet. Luettu 6.1.2011)

”Testaustekniikka sisältää ohjelmiston suorittamista löytääkseen siitä ohjelmointivirheitä, mutta tekniikka ei rajoitu ainoastaan siihen. Ohjelmistotestaamisella voidaan myös osoittaa, että (1) ohjelmisto/palvelu/tuote täyttää kaupalliset ja tekniset vaatimukset, (2) ohjelmisto toimii oletetusti ja (3) ohjelmisto voidaan implementoida halutuilla ominaisuuksilla.” (Wikipedia, Ohjelmiston testaaminen. Luettu 7.1.2011)

Ohjelmistotestauksen tärkeyden puolesta puhuu se, että ohjelmoinnin jälkeen ohjelma sisältää yleensä noin yhden virheen kahtakymmentä koodiriviä kohden. Kauemmin käytössä olleissa ohjelmissa arvioidaan olevan yksi virhe tuhatta koodiriviä kohden. Lisäksi on arvioitu, että noin 5% virheistä jää kokonaan löytämättä. (Haikala & Märijärvi 2004, 287-288)

4.1 Yksikkötestaus, hyödyt ja käyttö

Yksikkötesti on koodirakennelma, joka käsittelee toisen kokonaisuuden koodia ja tarkistaa, että se toimii oikein. Yleensä ohjelmoija, joka ohjelmoi testattavan koodikokonaisuuden, ohjelmoi myös yksikkötestin. Yksikkötesti voidaan ohjelmoida jopa ennen kuin varsinaisen ohjelman tekeminen on aloitettu, tästä tulee nimitys Test Driven Development. (Inside Microsoft Dynamics Ax 2009, 96)

Yksikkötestin tekemisessä varhaisessa vaiheessa on se hyöty, että se pakottaa ohjelmoijan miettimään tarkemmin, miten itse toteutus tehdään. Epäonnistunut yksikkötesti tarkoittaa, ettei ohjelma täytä kaikkia vaatimuksia. Yksikkötesti tulisi

suorittaa säännöllisin väliajoin, eikä sillä kuulu testata vain suuria kokonaisuuksia.

Yksikkötestiä tehdessä luodaan testiluokka, jota kutsutaan myös testitapaukseksi. Jokainen testitapaus sisältää useita metodeja, joiden idea on tarkistaa, että testattava ohjelma toimii halutulla tavalla.

Yksikkötestausta moititaan siitä, että ohjelmoijilta kuluu aikaa varsinaiselta tuottavalta työltä, koska he joutuvat käyttämään aikaa testitapauksien tekemiseen. Kuitenkin pidemmällä aikavälillä se tuo varmuutta tuotteeseen, koska ohjelmalliset virheet havaitaan ajoissa, mikä mahdollistaa rakenteen muutokset ennen kuin ohjelmaa on kehitetty liian pitkälle. Yksikkötestauksen pakottaman etukäteissuunnittelun avulla siis vältytään huonoilta väliaikaisratkaisuilta, jotka usein jäävät pysyviksi.

4.2 Yksikkötestauksen automatisointi

Yksikkötestaus on myös mahdollista automatisoida, mikä saattaa kuulostaa houkuttelevalta ajansäästämisen vuoksi. Kuitenkin tähän tarkoitettujen työkalujen on huomattu olevan hyvin epävakaita. Yleensä automaattisissa testiluokissa on myös paljon ylimääräistä, joka tekee niistä hitaita tai toimimattomia. Automatisoitu testaus ei myöskään sovi TDD:n periaatteisiin. (Litmanen W. 2010. Automaattisen toiminnallisen testauksen kehittäminen. Luettu 7.1.2011)

Automaattiset testit ovat kuitenkin toimineet joissain yrityksissä, mutta se on vaatinut hyvän käyttöliittymän ja pätevän henkilön päivittämään sitä käyttötarpeiden mukaan. Usein onkin niin, että testaajat saattavat joutua ohjelmoimaan enemmän kuin itse sovelluskehittäjät.

Microsoft Dynamics AX 2009 sisältää kuitenkin frameworkin SysTestin, joka mahdollistaa raskaimpien toiminnallisuuksien paikantamisen. SysTestistä kerrotaan tarkemmin myöhemmissä kappaleissa.

4.3 Testitapaukset

Testitapausta luodessa on luotava uusi luokka, joka periytyy SysTestCase-luokasta, joka on osa yksikkötestauksen rajapintaa. Hyviin nimeämiskäytäntöihin kuuluu nimetä testiluokka samalla nimellä kuin luokka, jota testataan, lisäämällä kuitenkin loppuun test-pääte. Tämä nimeämiskäytäntö kertoo yksikkötestausrajapinnalle, mistä testiluokasta se kerää tietoa. Jos nimeämiskäytäntö ei kuitenkaan sovi omiin käyttötarkoituksiin, nimi voidaan ohittaa testsElementName methodilla. Myös testsElementType metodi voidaan ohittaa, jos halutaan asettaa elementin tyyppi, josta tietoa kerätään. (Inside Microsoft Dynamics AX 2009, 97)

Onnistuneen testin luomiseksi on luotava vähintään yksi testimetodi. Jokaisen testimetodin nimen on alettava sanalla ’test’. Metodi ei saa palauttaa mitään, eikä sille saa välittää parametrejä. Jokaisessa testimetodissa on testattava, että haluttu olio, jota testataan toimii halutulla tavalla. Jos kutsutaan vain yhtä testimetodia onnistuneesti, Microsoft Dynamics AX 2009 ilmoittaa, että yksi testi ajettiin ja nolla testiä epäonnistui. Useita testimetodeja kutsuttaessa voidaan valita, pysähdytäänkö ensimmäiseen epäonnistuneeseen testiin vai mennäänkö eteenpäin.

Jos tarvitaan samoja alustuksia saman luokan useassa eri testimetodissa, niitä ei tarvitse alustaa jokaisessa metodissa erikseen, vaan voidaan luoda setUp-metodi. Kun testimetodia kutsutaan, se alustaa uuden testiolion, jonka jälkeen se kutsuu automaattisesti setUp-metodia. Mikäli halutaan vapauttaa muisti, voidaan kutsua tearDown-metodia. Näin ollen muisti on jälleen kokonaan käytössä toiseen testiin mentäessä, eikä varattu muisti väärennä seuraavan testin testituloksia.

Yksikkötestiframework mahdollistaa myös poikkeuksien testauksen. Mikäli poikkeusta odotetaan, framework voidaan asettaa odottamaan poikkeusta. Mikäli poikkeus tulee, yksikkötesti ilmoittaa testitapauksen epäonnistuneeksi. Frameworkille ilmoitetaan parmExceptionExpected([boolean, str])-metodia kutsumalla, että poikkeusta odotetaan. Mikäli parametri str ei vastaa täysin odotettua virheviestiä, testi epäonnistuu.

4.4 Testikokoelmat

Testikokoelmilla on kaksi käyttötarkoitusta. Toinen niistä on, että testitapaukset ja muut testikokoelmat saadaan lajiteltua, jolloin niiden löytäminen ja käyttäminen on helpompaa. Toinen tarkoitus on eristäminen. Eri testitapaukset ja testikokoelmat käsittelevät eri tietoja ja voi olla tarve eristää tiedot toisistaan. On myös suositeltavaa käyttää testiprojekteja testikokoelmien ryhmittelemiseksi. Microsoft Dynamics AX 2009:ssä on viisi eri testikokoelmaa, jotka kaikki tarjoavat oman eristystasonsa (Inside Microsoft Dynamics AX 2009, 101; Microsoft, Unit Test Framework. Luettu 6.1.2011)

SysTestSuite

SysTestSuite on vakio testikokoelma, eikä se tarjoa minkäänlaista eristystä.

SysTestSuiteCompanyIsolateClass

Koska Dynamics AX:ssa voidaan perustaa useita yrityksiä, on myös mahdollista, että testitapauksille luodaan oma yritys. Tämä kyseinen luokka luo tyhjän yrityksen ja ajaa jokaisen tämän testikokoelman sisällä tehdyn testin luodun yrityksen sisällä. Kun jokainen testimetodi on suoritettu, luotu yritys tuhotaan.

SysTestSuiteCompanyIsolateMethod

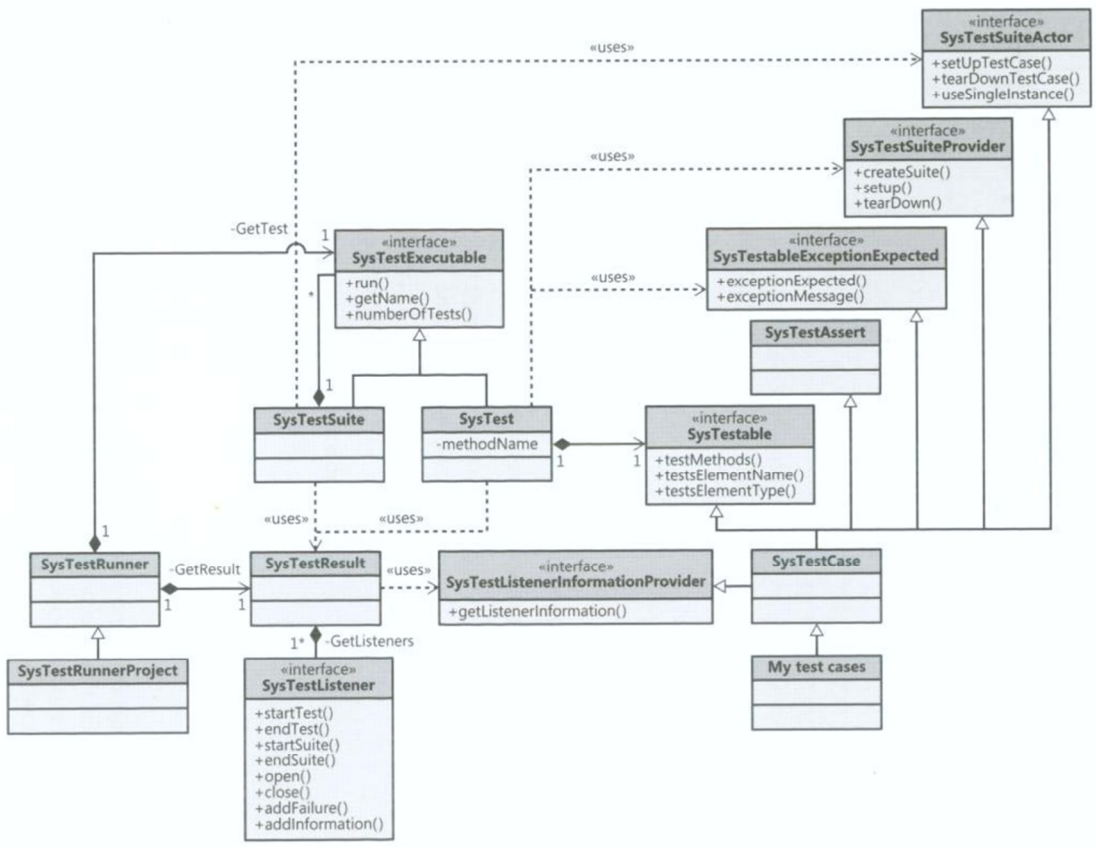
Tällä on lähes vastaava toiminnallisuus edellisen kanssa, mutta tämä testikokoelma luo tyhjän yrityksen jokaiselle metodille luokan sijaan. Kun testimetodit on ajettu, luodut yritykset tuhotaan. Tällä testikokoelmalla on korkein eristystaso ja sen suorittaminen on huomattavasti raskaampaa verrattuna muihin testikokoelmiin.

SysTestSuiteTTS

SysTestSuiteTTS testikokoelma pakkaa jokaisen testitapauksen transaktioon ja kun jokainen niistä on suoritettu, transaktio peruutetaan. Tämä on nopein kaikista testikokoelmista, mutta sillä on kaksi rajoitusta. Toinen on, ettei poikkeuksia voida käsitellä. Koska testikokoelma suoritetaan transaktion sisällä, se suoritetaan kerralla, eikä kesken tullutta poikkeusta voida havaita. Toinen rajoitus on testitapaukset, jotka täytyy kommitoida. Koska testikokoelma suorittaa jo itsessään kommitoinnin, se ei ymmärrä sitä enää testitapauksessa.

SysTestSuiteCompIsolateClassWithTts

Tämä testikokoelma on SysTestSuiteCompanyIsolateClassin ja SysTestSuiteTTS:n yhdistelmä, eli se luo testiyrityksen transaktion sisään ja kun testit on suoritettu, transaktio peruutetaan. Voisi siis ajatella että SysTestSuiteCompanyIsolateClass on SysTestSuiteTTS-testikokoelman sisällä.



KUVIO 2. UML-kaavio yksikkötestaus frameworkista. (Inside Microsoft Dynamics AX 2009, 107)

4.5 Testiprojektit

Kuten aikaisemmassa kappaleessa mainittiin, on suositeltavaa käyttää testiprojekteja testien lajitteluun. Dynamics AX 2009 omaa erittäin yksinkertaisen projektipuurakenteen, johon on helppoa lisätä projekteja, ryhmiä sekä elementtejä sen sisälle. On myös mahdollista luoda joko yksityinen tai julkinen projekti. Mikäli on luotu julkinen projekti, on suositeltavaa käyttää versionhallintaa päällekkäisyyksien estämiseksi. (Inside Microsoft Dynamics AX 2009, 103)

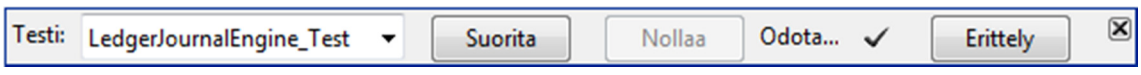
Testiprojektit voivat myös sisältää viittauksia toisiin projekteihin, mikä sallii usean eri kehitystiimin käyttävän samaa projektia.

4.6 Testityökalupalkki ja koodin kattavuus

Testityökalupalkki nopeuttaa Microsoft Dynamics AX:n testien suorittamista entuudestaan, koska työkalupalkista voidaan valita suoraan ajettava testi. Sen avulla pystytään myös ajamaan useita eri testejä peräkkäin, eikä testiluokkia tarvitse aina etsiä testiprojekteista. (Inside Microsoft Dynamics AX 2009, 104)

4.6.1 Testityökalupalkki

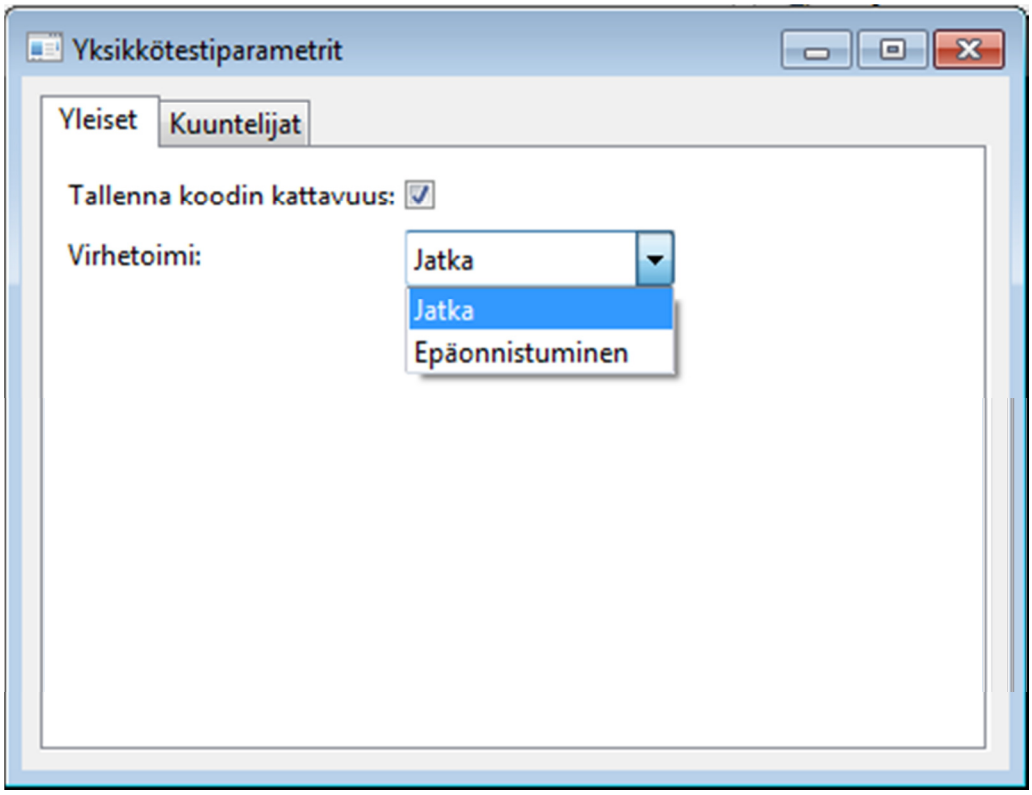
Testityökalupalkin tarkoituksena on nopeuttaa testitapauksen, testiluokan, testikokoelman tai testiprojektin ajoa. Testin nimen kirjoittaminen kenttään riittää, jolloin testi voidaan suorittaa ja nähdä sen tiedot. Tästä syystä on tärkeää nimetä testit kuvainnollisesti, jottei kallisarvoista aikaa kuluisi testien etsimiseen. Ajon jälkeiset tiedot selitetään tarkemmin myöhemmässä kappaleessa.



KUVIO 3. Yksikkötestauksen testityökalupalkki.

4.6.1 Koodin kattavuus

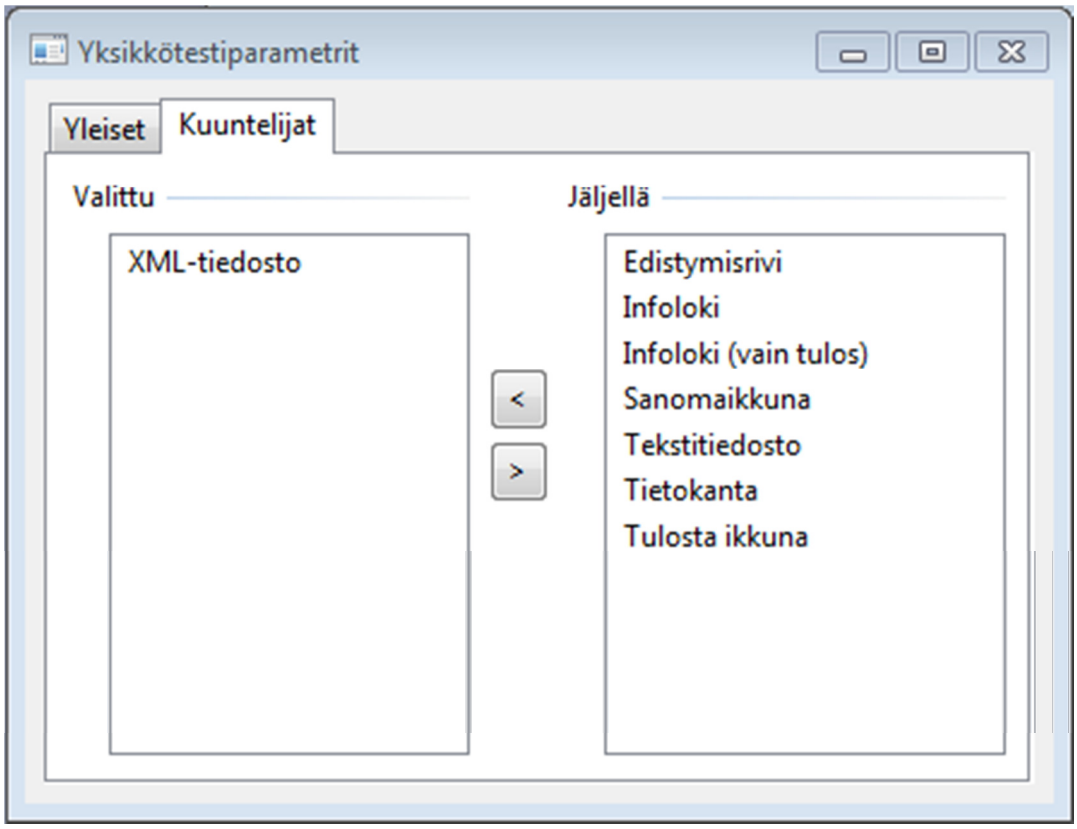
Yksikkötestirajapinnan avulla voidaan myös kerätä tietoa koodin kattavuudesta, joka asetetaan parametreistä. Se pystyy kertomaan kuinka monta prosenttia on testattu halutusta ohjelmasta. Sen avulla voidaan myös selvittää, mitkä sovelluksen osa-alueet eivät tule testatuksi muiden testitapausten avulla. Koodin kattavuus saadaan asetettua päälle yksikkötestausparametreistä. Tietenkään tiedon keruun lisääntyessä, lukemat eivät ole niin totuudenmukaiset kuin ilman koodinkattavuus-rajapintaa. Onkin suositeltavaa tehdä performanssitestit ilman kyseistä parametriä ja suorituskyvyn selvittyä ajaa sovellus uudestaan testin läpi. Yksikkötestausparametreistä voidaan myös valita virhetoimi, eli mitä tehdään virheen sattuessa. Vaihtoehtoina on joko suorittaa testi loppuun, tai keskeyttää se ensimmäiseen virheeseen.



KUVIO 4. Yksikkötestiparametrit.

4.7 Testikuuntelijat

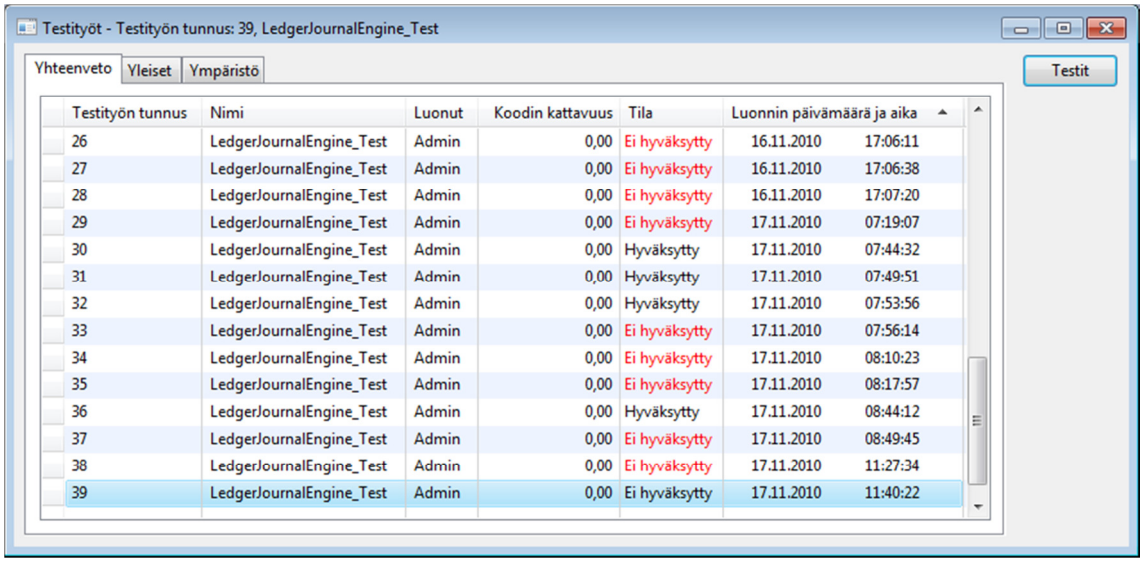
Kun ajetaan testitapausta tai testikokoelmaa voidaan Microsoft Dynamics AX 2009:n kytkeä kuuntelijoita. Jokainen kuuntelija tuottaa erilaisen tulosteen. Tulosteet voivat olla muun muassa tekstitiedosto, xml-tiedosto, tietolomake, tulostelomake tai jopa etenemispalkki. Kuuntelijat kytketään päälle yksikkötestausparametreistä. (Inside Microsoft Dynamics AX 2009, 105)



KUVIO 5. Testikuuntelijat.

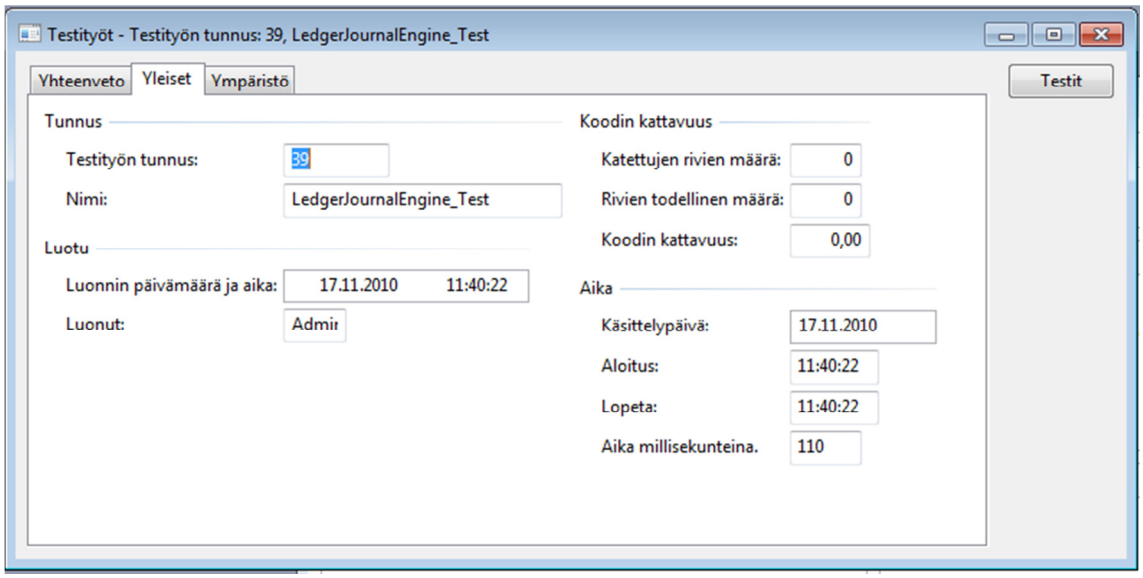
4.8 Testaushistoria

Microsoft Dynamics AX 2009 kerää lokin jokaisesta testistä, joka on tehty. Testeistä selviää testin järjestysnumero, nimi, tekijä, koodin kattavuus, testin tila sekä muita hyödyllisiä tietoja, kuten ympäristö, jossa testi on ajettu.

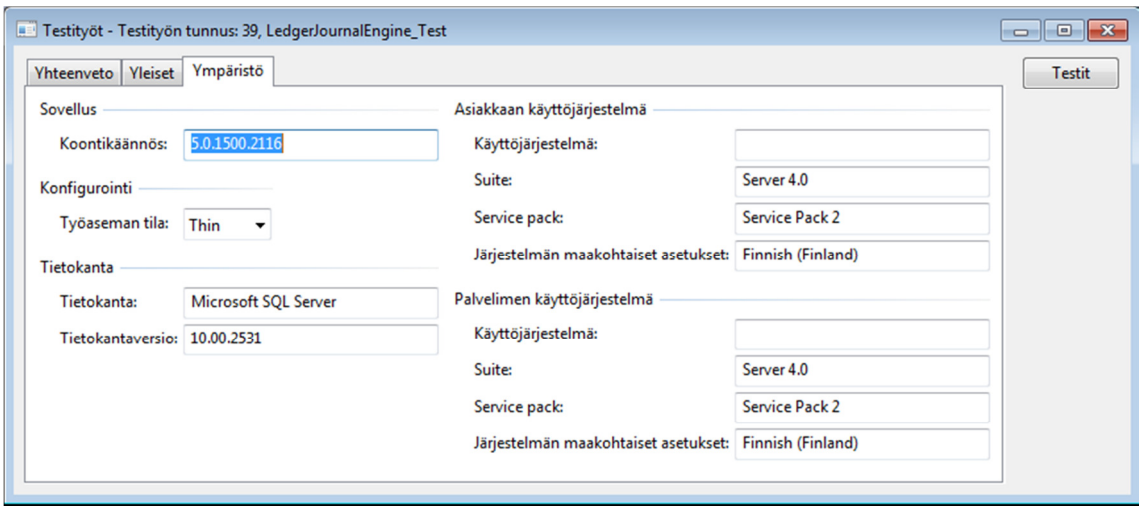


Testityön tunnus	Nimi	Luonut	Koodin kattavuus	Tila	Luonnin päivämäärä ja aika
26	LedgerJournalEngine_Test	Admin	0,00	Ei hyväksytty	16.11.2010 17:06:11
27	LedgerJournalEngine_Test	Admin	0,00	Ei hyväksytty	16.11.2010 17:06:38
28	LedgerJournalEngine_Test	Admin	0,00	Ei hyväksytty	16.11.2010 17:07:20
29	LedgerJournalEngine_Test	Admin	0,00	Ei hyväksytty	17.11.2010 07:19:07
30	LedgerJournalEngine_Test	Admin	0,00	Hyväksytty	17.11.2010 07:44:32
31	LedgerJournalEngine_Test	Admin	0,00	Hyväksytty	17.11.2010 07:49:51
32	LedgerJournalEngine_Test	Admin	0,00	Hyväksytty	17.11.2010 07:53:56
33	LedgerJournalEngine_Test	Admin	0,00	Ei hyväksytty	17.11.2010 07:56:14
34	LedgerJournalEngine_Test	Admin	0,00	Ei hyväksytty	17.11.2010 08:10:23
35	LedgerJournalEngine_Test	Admin	0,00	Ei hyväksytty	17.11.2010 08:17:57
36	LedgerJournalEngine_Test	Admin	0,00	Hyväksytty	17.11.2010 08:44:12
37	LedgerJournalEngine_Test	Admin	0,00	Ei hyväksytty	17.11.2010 08:49:45
38	LedgerJournalEngine_Test	Admin	0,00	Ei hyväksytty	17.11.2010 11:27:34
39	LedgerJournalEngine_Test	Admin	0,00	Ei hyväksytty	17.11.2010 11:40:22

KUVIO 6. Yhteenvetonäkymä toteutetuista testitapahtumista.



KUVIO 7. Yleisiä testitapahtuman suoritustietoja



KUVIO 8. Ympäristön tiedot suoritetusta testitapahtumasta.

4.9 Johtopäätökset

Yksikkötestaustyökalut auttavat varmistamaan, että koodi toimii halutulla tavalla ja TDD:n avulla päästään haluttuihin lopputuloksiin. Testityökalut kuitenkin vaativat paljon ohjelmakoodia ja määritelmien tulee olla hyvin tehty jo alusta alkaen, jottei arvokasta ohjelmointiaikaa kulu testiluokkien korjaukseen ja parantamiseen. On siis mietittävä tarkkaan, säästetäänkö loppujen lopuksi aikaa testityökalujen avulla. Vaikkakin niiden ansiosta päästään haluttuun lopputulokseen suoraviivaisesti, niiden kautta lopputulokseen pääseminen voi olla pidempi tie, kuin mitä se olisi ilman niiden käyttöä. Toisaalta ilman kyseisiä työkaluja joudutaan varmasti useaan kertaan korjaamaan jo olemassa olevaa ohjelmistokoodia.

Yksi hyödyllisin ominaisuus testityökaluissa on performanssitestit. Niiden avulla voidaan selvittää kauanko prosessiin kuluu aikaa ja etsiä kohdat, jotka voitaisiin mahdollisesti ohjelmoida tehokkaammiksi. Performanssitesteistä kerrotaan tarkemmin seuraavassa kappaleessa.

5 PERFORMANSSITESTI

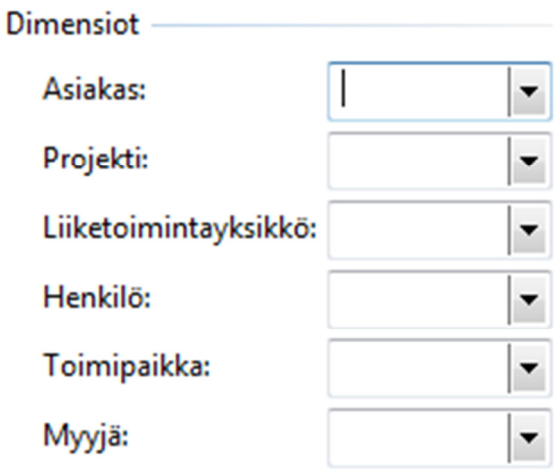
Kuten jo aikaisemmin mainittiin, Microsoft Dynamics AX 2009:n yksikkötestaustyökalu sisältää muun muassa performanssitestin. Yksinkertaisella testiprojektilla nähdään kuinka paljon enemmän aikaa kuluu kirjaustapahtuman tekemiseen aineiston kasvaessa ja vaikuttavatko virheet kirjausaikaan. Pienessä aineistossa on vain kolme dimensiota käytössä eikä siinä ole virheitä, kun taas suuressa aineistossa on yhdeksän dimensiota ja aineisto sisältää vanhentuneita dimensioita. Näin ollen nähdään kuinka paljon tapahtumien kirjaus hidastuu ja voidaan myös havaita kasvaako kirjauksen kesto suhteessa virheiden määrään. Koodin kattavuus, kuuntelijat sekä muut parametrit ovat pois käytöstä, etteivät ne vääräntäisi testituloksia, kuten aikaisemmin mainittiin kappaleessa 3.

5.1 Dimensiot ja tehtävät

Dimensiot ovat keskeinen osa Dynamics AX:n raportointia ja ne voidaan määrittää esimerkiksi tehtävien avulla halutuille kirjauskansion riveille.

Dimensiot

Dimensiot voivat olla haluttua tietotyyppiä ja niitä voi olla Microsoft Dynamics AX 2009:ssä korkeintaan kymmenen, vakiona niitä on kolme. Dimensiot ovat raportointitasoja ja ne auttavat selvittämään eri perspektiiveistä minkälaisesta tapahtumasta on kyse, mihin se liittyy ja helpottavat jäljittämään, mistä tapahtuma on lähtenyt liikkeelle. Dimensioita voisi siis ajatella ryhmittelyehtoina. Kuviossa 9 on esimerkki dimensioista.



KUVIO 9. Esimerkki dimensioista.

Tehtävät

Tehtävät ovat globaaleja funktioita, joita suoritetaan runtime-ympäristössä. Niitä käytetään usein testauksessa tai suorittamaan haluttu toimenpide. Ne ovat kuin pieniä ajettavia ohjelmia. Tässä tapauksessa tehtävällä luodaan kirjauskansiolle rivejä, koska käsin luomisessa kuluisi aikaa useita tunteja. Tehtävällä se onnistuu hetkessä.

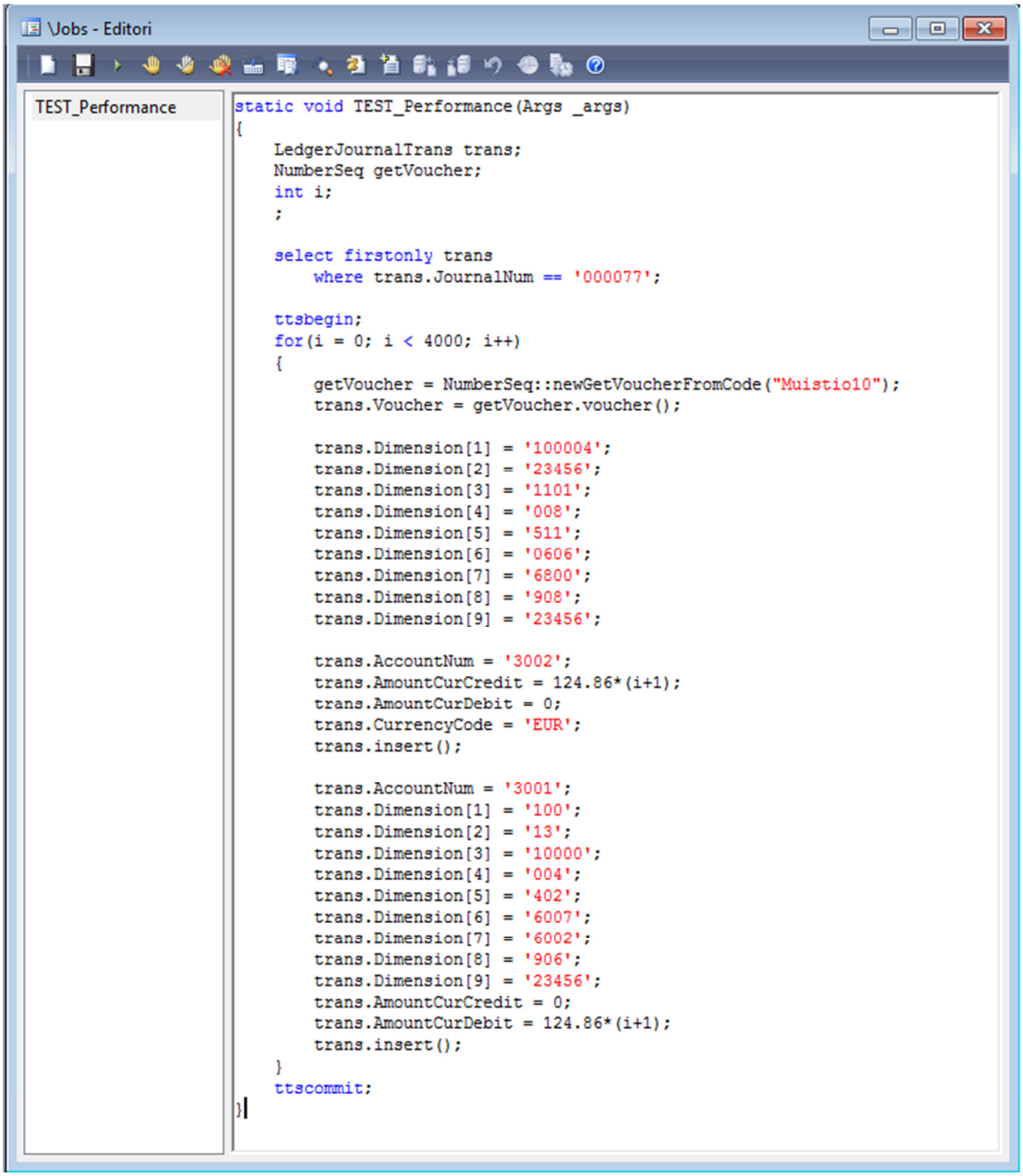
5.2 Testiohjelma

Testiohjelman tulisi olla mahdollisimman monikäyttöinen, joten kirjauskansion rivien luonti tehdään tehtävällä seuraavassa esimerkissä.

Kirjauskansion rivien luonti

Kuviossa 10 on luotu olio LedgerJournalTrans-luokasta ja sille on määritelty kirjauskansio, jonne halutut rivit luodaan. Kirjauskansio on luotu manuaalisesti. Tositesarjat (voucher) on määritelty haettavaksi Muistio10:n takaa, joka on itse määritelty. Tässä tapauksessa kaikki dimensiot ovat string-tyyppisiä. Tehtävä luo debit-tyyppisen kirjausrivin asiakkaalle 3002 ja vastaavasti kredit-tyyppisen

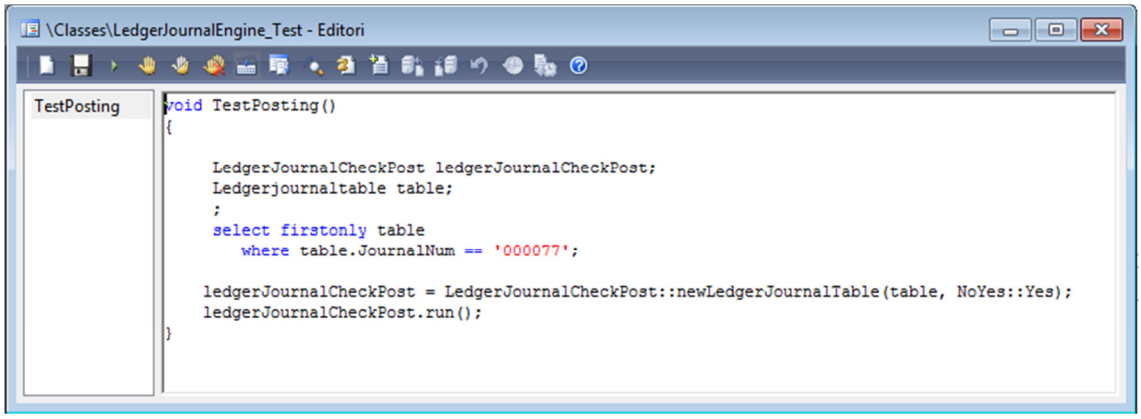
asiakkaalle 3001. Rivejä luodaan kaiken kaikkiaan 4000 kappaletta, eli kaiken kaikkiaan tulee 2000 eri tositenumeroa, koska tositenumerolle on oltava aina pari. Tehtävä ajetaan vihreästä nuolesta, joka sijaitsee kuvion vasemmassa yläreunassa tai painamalla F5.



KUVIO 10. Kirjausrivien luomiseen käytetty tehtävä.

Kirjaaminen

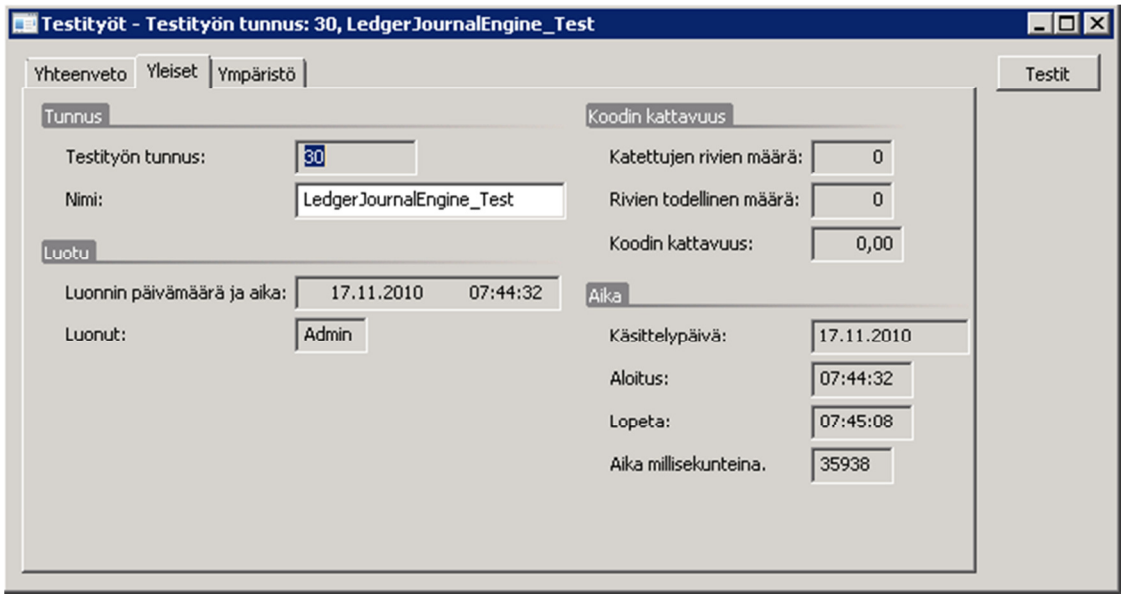
Luokassa LedgerJournalEngine_test oleva TestPosting()-metodi etsii halutun kirjauskansion, jonka jälkeen standardi Dynamics AX-luokka LedgerJournal-CheckPostista luodaan olio, johon sijoitetaan siellä olevat tapahtumat ja se ajetaan, mikä hoitaa kirjauksen. Testiluokka ajetaan yksikkötestaustyökalusta kirjoittamalla siihen Testiluokan nimi ja painamalla ”suorita”. Kun testiohjelma on suoritettu, aukeaa automaattisesti dialogi, joka kertoo testitulokset.



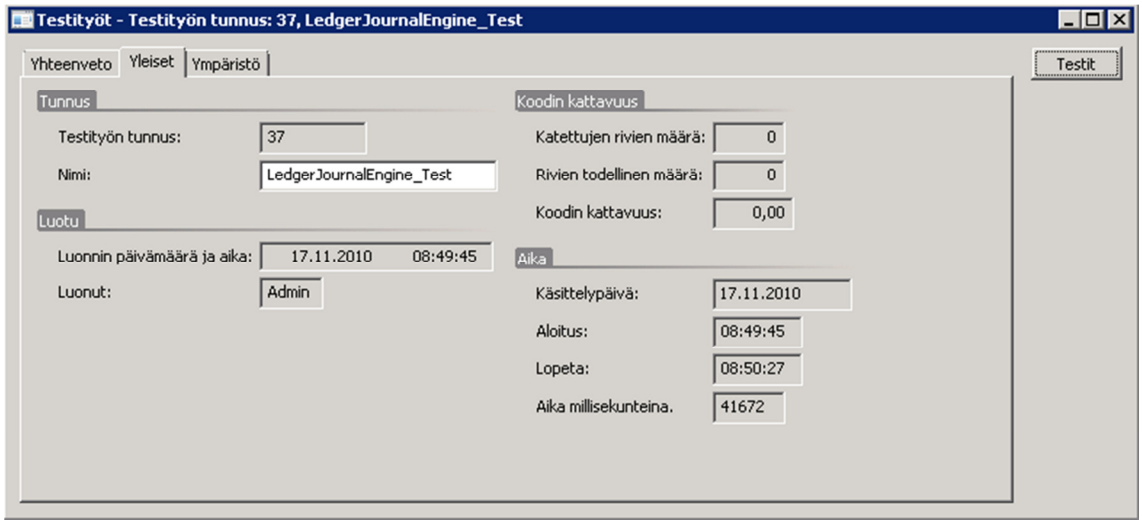
KUVIO 11. Ajettava testiluokka.

5.3 Testitulokset

Kuvioiden 12 ja 13 testituloksista voidaan havaita, että suuren aineiston usean dimension kirjaustapahtumien kirjausprosessi ei hidastu suhteessa muutaman rivin tai muutaman dimension kirjaukseen, eikä virheiden olemassaolo myöskään vaikuta suorituskykyyn. 4000 rivin kirjaus, ilman virheitä, kolmella dimensiolla kesti 36 sekuntia ja kun aiheutettiin virheitä vanhentuneilla dimensioilla ja otettiin 10 dimensiota käyttöön, aikaa kului 42 sekuntia.



KUVIO 12. 4000:n rivin kirjausaika kolmella dimensiolla ilman virheitä.



KUVIO 13. 4000:n rivin kirjausaika kymmenellä dimensiolla virheiden kanssa.

5.4 Johtopäätökset

Vaikkakin dimensiomäärää kasvatettiin ja virheitä lisättiin, kirjaustapahtuman kesto ei kasvanut merkittävästi. Täytyy ottaa myös huomioon, että jokaisesta virheestä Microsoft Dynamics AX 2009 kirjoittaa lokin lokitiedostoihin ja tulostaa lopuksi tekstiruudun, jossa on lueteltu virheet. Tekstiruutuun mahtuu enimmillään 1000 ilmoitusta ja kyseisessä testissä niitä oli yli tämän määrän. Stringin käsittely on myös erittäin raskas operaatio jokaiselle ohjelmointikielelle, joten kuuden sekunnin lisääntynyt kirjausaika on hyvin vähäinen.

6 YHTEENVETO

Tutkintotyön tavoitteena oli esitellä Microsoft Dynamics AX 2009–toiminnanohjausjärjestelmää sekä tähän liittyviä yksikkötestauksen toiminnallisuuksia, parametreja ja käyttöä tässä järjestelmässä, sekä esimerkin avulla esittää yksikkötestauksen hyödyntämismahdollisuuksia kyseisessä järjestelmässä.

Microsoft Dynamics AX 2009 on kehittynyt huomattavasti muutamassa vuodessa ja se on suosittu ERP-järjestelmä maailmassa, minkä seurauksena jatkokehitys on välttämätöntä. Microsoftin antamien tiedotteiden mukaan seuraavan sukupolven edustaja, Microsoft Dynamics AX 6 näkee päivänvalon vuoden 2011 aikana. Dynamics AX 6 tuo uudistuksia varsinkin käyttöliittymään. Tietyille toimialoille, kuten kaupan alalle, media-alalle ja julkiselle sektorille tullaan Microsoftin mukaan panostamaan uudistettua versiota julkaistaessa.

Dynamics AX:n käyttäjien räjähdysmäinen kasvu ja sitä kautta kasvaneet vaatimukset ja asiakkaiden yksilöimistoiveet ovat luoneet myös testaukselle haastavan ympäristön, jotta iteraatiokierroksilta vältyttäisiin. Tässä jatkuvasti kehittyvässä ja uudistuvassa toiminnanohjausjärjestelmässä tuotetaan moninkertainen määrä muutoksia verrattuna useisiin stabiileihin toiminnanohjausjärjestelmiin, jolloin myös testauksen on oltava jatkuvaa.

Nykypäivänä ohjelmistokehityksessä on huomioitavissa siirtymävaihe niin kutsuttuja ketteriä menetelmiä (Agile Methods) kohti, joissa lyhyehköissä jaksoissa tapahtuvat iteraatiot ovat hiljalleen syrjäyttämässä perinteisen vesiputousmallin. Ohjelmistokehityksessä iteraatio-mallilla toimien, saattaa ohjelmistoon tulla suuriakin muutoksia lyhyellä aikavälillä, mikä asettaa haasteita yksikkötestaukselle, joka ei täysin sovellu projekteihin, joissa muutoksia tapahtuu nopealla vauhdilla. Tämä johtuu siitä, että perinteisessä vesiputousmallissa ohjelmiston sisältö ja toivotut toiminnallisuudet on määritelty ennen projektin aloittamista hyvinkin tarkasti, eikä niistä useimmiten projektin edetessä voida poiketa. Ketterissä menetelmissä, joissa projektiin kohdistuvat iteraatiokierrokset kestävät tyypillisimmin

1-2 viikkoa on normaalia, ettei määrittelyä tehdä kovinkaan kattavasti tarkoituksena saada nopeasti toimiva ohjelmistoversio esiteltäväksi. Eli yleensä jo ensimmäisen ”sprintin” jälkeen tulisi olla ohjelmistosta toimiva versio, joskin hyvin vähillä toiminnallisuuksilla. Tämän jälkeen päätetään mihin suuntaan kyseisen ohjelmiston kohdalla suuntaudutaan ja saattaa jopa käydä niin, että ensimmäisen kierroksen aikana tulee ilmi asioita, joiden takia koko projektirakenne ja tavoitteet voivat muuttua.

Edellä mainittu muutos tavassa toteuttaa ohjelmistoprojekteja, on monimutkaistanut myös yksikkötestauksen käytön yleisesti ohjelmistokehityksessä, sekä myös Microsoft Dynamics AX- toiminnanohjausjärjestelmää koskien. Nykypäivän vaativat asiakkaat eivät myöskään useimmiten tyydy pelkkään pakettiratkaisuun, jolloin asiakkaiden toiveiden mukaan räätälöity järjestelmä vaikeuttaa myös yksikkötestausta, jossa testi käytännössä luodaan ennen kuin varsinainen ohjelmisto tuotetaan.

Yksikkötestaus on nykypäivän haasteistaan huolimatta edelleen yksi toimivimmista tavoista varmistaa tuotetun ohjelmiston toimivuus, mutta tapa toteuttaa yksikkötestausta on muuttunut ja tulee vielä tulevaisuudessa muuttumaan lisää. Vaikkakin Microsoft Dynamics AX on jatkuvan kehityksen kohteena, pysyvät sen perustoiminnallisuudet ja ohjelmiston toimintalogiikka lähes muuttumattomana, jolloin yksikkötestausta voidaan edelleen laajamittaisesti hyödyntää, eivätkä pienimuotoiset yksittäiset asiakaskohtaiset räätälöinnit välttämättä estä sen käyttöä.

Nykypäivänä ohjelmistokehityksen prosessit, teknologiat ja ohjelmointikielet itsessään ovat hioutuneet niin toimiviksi kokonaisuuksiksi, että kokenut ohjelmistotestaaja pystyy käytännössä testaamaan koodia sitä mukaa, kun sitä valmistuu. Tällöin eritoten räätälöintitapauksissa yksikkötestein voidaan testata laajempia kokonaisuuksia, mutta räätälöityjä muutoksia testattaessa kokenut ja osaava testaaja on korvaamaton.

LÄHTEET

Inside Microsoft Dynamics AX 2009. The Microsoft Dynamics AX team. 2009.

Microsoft Dynamics AX. Luettu 4.1.2011.
http://www.microsoft.com/dynamics/fi/fi/products/ax-overview.aspx

Unit Test Framework. Luettu 6.1.2011.
http://msdn.microsoft.com/en-us/library/aa874515.aspx

Wikipedia. 2011. Toiminnanohjausjärjestelmä. Luettu 3.1.2011.
http://fi.wikipedia.org/wiki/ERP_(johtamisjärjestelmä)

Wikipedia. 2011. Ohjelmiston testaaminen. Luettu 7.1.2011.
http://fi.wikipedia.org/wiki/Ohjelmiston_testaaminen

Haikala & Märijärvi 2004. Ohjelmistotuotanto.

Wikipedia. 2011. Microsoft Dynamics AX. Luettu 5.1.2011.
http://en.wikipedia.org/wiki/Microsoft_Dynamics_AX

Testauksen tavoitteet. Luettu 6.1.2011.
http://www.oamk.fi/sbc/testaus/testauksen_tavoitteet.htm

Litmanen, W. 2010. Automaattisen toiminnallisen testauksen kehittäminen. Oulun seudun ammattikorkeakoulu. Opinnäytetyö. Luettu 7.1.2011.
https://publications.theseus.fi/bitstream/handle/10024/21622/Litmanen_Wille.pdf